2017 International Conference on
Unmanned Aircraft Systems (ICUAS)
June 13-16, 2017, Miami, FL, USA

Dancing UAVs: Using Linear Programming to Model Movement Behavior with Safety Requirements

Hoang Tung Dinh
imec-DistriNet
KU Leuven
3001 Leuven, Belgium
hoangtung.dinh@cs.kuleuven.be

Mario Henrique Cruz Torres
imec-DistriNet
KU Leuven
3001 Leuven, Belgium
mariohenrique.cruztorres@cs.kuleuven.be

Tom Holvoet
imec-DistriNet
KU Leuven
3001 Leuven, Belgium
tom.holvoet@cs.kuleuven.be

***Abstract*—In this paper we present the use of linear programming to systematically create control software for choreographed UAVs. This application requires the control of multiple UAVs where each UAV follows a predefined trajectory while simultaneously maintaining safety properties, such as keeping a safe distance between each other and geofencing. Modeling and incorporating safety requirements into the movement behavior of UAVs is the main motivation of our research. First, we describe an approach where the movement behavior of each UAV is formulated as a linear program. Second, we compare and analyze two different modeling techniques to implement the safe distance and geofencing requirements. Our approach was validated by doing experiments with Parrot Bebop UAVs. Besides being tested in the laboratory, our approach was validated in real life conditions in more than 30 performances of a dance show where five UAVs perform choreographed movements as part of the show introduction.**

I. Introduction

Recently UAVs have been attracting considerable attention in both academia and industry. A particularly demanding UAV application is performing choreographed movements [1]–[4], where a group of UAVs follow predefined trajectories while still guaranteeing safety constraints such as keeping a safe distance and geofencing.

Incorporating safety requirements into the choreographed movement of UAVs *at runtime* is challenging. It is not feasible to know in advance all the variations that may happen in the environment of the UAV at runtime. The control software of the UAV needs to cope with the dynamics of the UAV surroundings while guaranteeing a number of safety requirements. Creating a software which copes with such requirements individually is complex and it becomes even more complex when the software has to cope with many safety requirements simultaneously. Besides being correct, the software also has to be efficient since it has to be executed in a real time manner. In the literature, to the best of our knowledge, safety for choreographed UAVs is only considered in an offline manner [2], [4] by designing collision-free trajectories. There is a need for a systematic way to develop software that can handle the integration between movement behavior and safety requirements in a consistent and predictable manner during execution.

In this paper we propose an approach that systematically accommodates concerns such as:

- choreographed movement
- safety requirements such as keeping a safe distance and geofencing
- low level control of the UAV attitude
- low computational cost

Our approach is rooted on *constraint-based programming*, where the behavior of a robot is formulated as a constrained optimization problem [5]–[9]. We create a linear constrained optimization model (a linear program) that represents the desired movement behavior and safety constraints imposed on the UAV. The linear program in our approach can be solved efficiently by existing open-source and commercial solvers, enabling our approach to be used to control the UAVs at runtime.

The contributions of this paper are two-fold. First, we describe our approach to develop software to control multiple indoor choreographed UAVs using linear programming. We show how to model the runtime movement behavior of each UAV, taking into account safety requirements, as a linear program. The control of the UAV group is decentralized. Each UAV solves its own linear program, improving system's robustness and scalability. Second, we compare and analyze two different modeling techniques to represent safety requirements concerning keeping a safe distance and geofencing. The approach is validated in a university laboratory and at a dance show with five Parrot Bebop UAVs[1] performing a choreography on stage.

This paper is organized as follows. Section II discusses related work. Section III gives an overview of the scenario. Section IV details the linear program used in our approach.

[1]Parrot Bebop UAV is a lightweight commercial UAV platform that counts with a 14 Mega pixels fisheye lens camera and a simple to use Software Development Kit (SDK).

978-1-5090-4494-8/17/$31.00 ©2017 IEEE

Section V presents and discusses our validation results. Finally, Section VI draws conclusions and details possible future work.

II. Related work

Recent studies on using UAVs to perform choreographed movements [1]–[4] focus on designing trajectories and ensuring safety offline. Such trajectories are validated through simulations in order to check their feasibility. At runtime, the only task of the UAVs is to follow their predefined trajectories without any efforts to guarantee their safety requirements. Our work differs from these in that we focus on maintaining safety requirements at runtime, assuming that choreographed trajectories are given a priori.

Modeling robot movement behavior as a constrained optimization problem is also used in robot motion planning [10]–[12] and Model Predictive Control (MPC) [13], where the trajectory and control sequence of the robot are computed and optimized up to a finite horizon in the future based on the robot's dynamic and kinematic models. These techniques are not fast enough to provide reaction in critical situations [14]. Nevertheless, they can be integrated with reactive controllers to provide both high-level planning and low-level reactive capabilities [11], [15]. In this paper we focus on developing a reactive controller for UAVs. Details on coupling planning and reactive control are beyond the scope of this paper.

Constraint-based programming software frameworks for developing reactive robot behavior such as *Stack of Tasks* (SoT) [6] and *instantaneous Task Specification using Constraints* (iTaSC) [5], [16], [17] are already used in robotics programming. Although constraint-based programming has received much attention in the robotics field, there are surprisingly few examples of its application to working UAV systems. Different from robot arm manipulators [18] and humanoid robots [19], UAVs, especially multi-rotor UAVs, have simpler kinematic and dynamic models [20], which simplifies modeling and makes constraint-based programming a promising approach to model the movement behavior of UAVs.

Most related to our work is the work of Verbeke et al. [8]. They demonstrate an iTaSC controller that limits the movement of a UAV steered by a pilot. Their least-squares model is used in a reactive and passive manner in order to achieve requirements such as keeping the UAV at a safe distance to walls and dynamic obstacles. Their safe distance constraint is passive in the sense that the constraint is only activated when the UAV has already violated a safe distance to an obstacle. Our work differs from [8] in two aspects. First, we model the movement behavior of each UAV in our application using linear programming, which allows more freedom in modeling than using least-squares. Second, we propose a way to model the safe distance requirement so that the model actively takes into account the safe distance all the time, even when the safe distance is not being violated.

III. Scenario

Our scenario involves multiple UAVs performing choreographed movements. Each UAV follows a predefined trajectory. The trajectory $\mathbf{x_d}(t)$ of each UAV is a continuous function over time.

$$\mathbf{x_d}(t) = [x_d(t) \quad y_d(t) \quad z_d(t) \quad \psi_d(t)] \tag{1}$$

where $x_d(t)$, $y_d(t)$, $z_d(t)$ are the desired position of the UAV in three-dimensional space at time t and $\psi_d(t)$ is the desired yaw angle of the UAV at time t.

The task of each UAV is to follow its trajectory as closely as possible while still guaranteeing safety requirements. Three safety requirements need to be satisfied at runtime:

1) *Safe distance:* The UAVs must avoid colliding with each other. That is, each UAV must maintain a safe distance of at least d_{safe} from other UAVs.
2) *Geofencing:* The UAVs must always stay within a safe region. The safe region is defined by six planes forming a box.
3) *Only move if visual localization information is available:* Each UAV only moves if there is up-to-date visual localization information available. If there is no visual localization information received within a predefined duration, the UAV must hover until it receives new visual localization information.

There is another important safety aspect concerning the system as a whole. As there are multiple UAVs in the system, it is simpler and safer to have a homogeneous and decentralized control software. In our system, there is an instance of control software per UAV, which only exchange location information with other control software instances. That way, the failure of a UAV's control software does not impact the other UAVs in the system. It also enables the system to be executed on multiple computers while simplifying the software development.

In our scenario, we use the Parrot Bebop UAVs because of their sturdiness and affordable price. A Parrot Bebop UAV comes with IMU, ultrasonic sensor, vertical optical flow camera, fisheye camera, GPS and air pressure sensor. Since they are operated indoors, GPS information is not available. A Parrot Bebop UAV is equipped with an onboard flight controller that accepts control command of roll angle, pitch angle, vertical velocity and yaw rotational velocity. Although we use Parrot Bebop UAVs as the target implementation, our modeling techniques in this paper are general enough to be applied to different UAV platforms.

IV. Linear programming controller

In this section we present how we model the movement behavior of a UAV as a linear program.

Figure 1 illustrates the use of *linear programming controllers (LPCs)*. Each UAV is controlled by an independent LPC. At each time step (control loop), a LPC gets (1) the current estimated pose and velocity of its UAV and (2) the current positions of other UAVs as inputs to calculate the control command for the UAV. Note that the only information shared between the UAVs is their current position.

In the rest of this section we detail the linear program implemented in each LPC. The constraints in Section IV-A, IV-B and IV-C are general enough to be applied to different

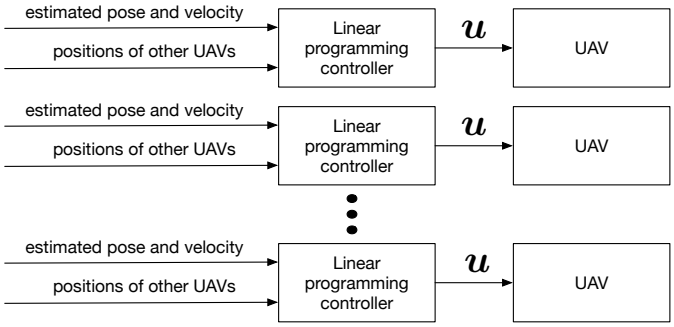


Fig. 1: Decentralized control. Each UAV is controlled by a linear programming controller (LPC). At each time step, a LPC takes (1) the current estimated pose and velocity of the UAV and (2) the current positions of other UAVs as the inputs to generate control command $\boldsymbol{u}$ for the UAV.

multi-rotor UAV platforms. The constraints in Section IV-D and IV-E are specific for the Parrot Bebop UAVs. We use $\boldsymbol{bold\quad italic}$ symbols to denote matrices holding decision variables and $\mathbf{bold-upright}$ symbols to denote matrices holding values.

A. Velocity controller

We model the velocity $\boldsymbol{v}$ of the UAV in the next time step in a global reference frame as decision variables. The velocity $\boldsymbol{v}$ will be used to derive the control command $\boldsymbol{u}$ (Section IV-E).

$$\boldsymbol{v} = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{\psi}]^T \tag{2}$$

Since the first goal of the LPC is to follow the desired trajectory as closely as possible, we employ a proportional-derivative (PD) velocity controller. At each control loop, the LPC gets the current estimated pose $\mathbf{x_e}$ and velocity $\mathbf{v_e}$ of the UAV from the localization software module. Let $\mathbf{x_d}$ be the current desired pose and $\mathbf{v_d} = \dot{\mathbf{x}}_\mathbf{d}$ be the desired velocity (in the global reference frame) given by the trajectory (see Section III). The PD controller computes $\mathbf{v_{pid}}$ using the following equation.

$$\mathbf{v_{pid}} = \mathbf{K_P^v}(\mathbf{x_d} - \mathbf{x_e}) + \mathbf{K_D^v}(\mathbf{v_d} - \mathbf{v_e}) \tag{3}$$

$\mathbf{v_{pid}}$ is the velocity to follow the predefined trajectory. $\mathbf{v_{pid}}$ does not take into account any safety requirements. Since the UAV needs to follow the trajectory as closely as possible while still complying with safety requirements, we define the objective function of the linear program such that it minimizes the difference between $\boldsymbol{v}$ and $\mathbf{v_{pid}}$. Concretely, the objective function is the $l1$-norm of the absolute difference $\Delta\boldsymbol{v}$ between $\mathbf{v_{pid}}$ and $\boldsymbol{v}$.

$$\text{minimize } ||\Delta\boldsymbol{v}||_1 \tag{4}$$

where

$$\Delta\boldsymbol{v} = |\boldsymbol{v} - \mathbf{v_{pid}}| \tag{5}$$

Since Equation (5) is non-linear, we convert it to two equivalent linear constraints.

$$\begin{aligned} \Delta\boldsymbol{v} &\geq \boldsymbol{v} - \mathbf{v_{pid}} \\ \Delta\boldsymbol{v} &\geq \mathbf{v_{pid}} - \boldsymbol{v} \end{aligned} \tag{6}$$

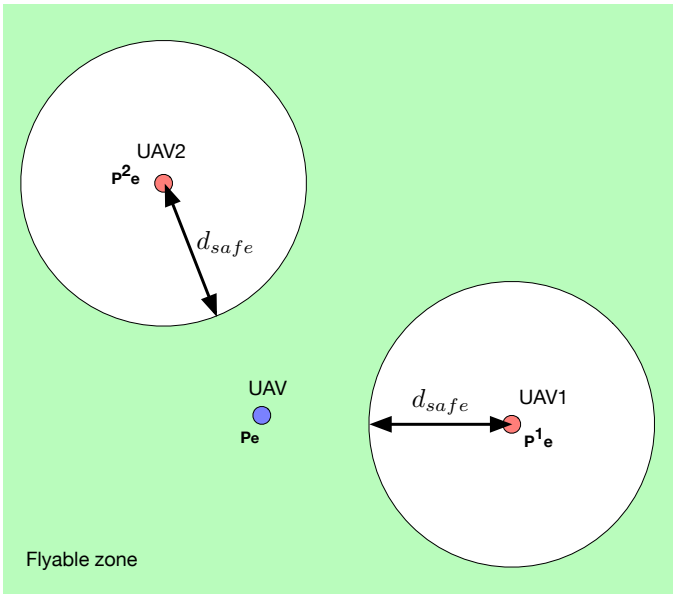


Fig. 2: The basic form of the safe distance constraint. The current UAV (blue dot) needs to keep a distance of at least d_{safe} to UAV1 and UAV2. The flyable area forms a non-convex region.

B. Safe distance

The first safety requirement is that a UAV must always keep a safe distance of at least d_{safe} from other UAVs. We present two different ways to model this safety requirement: *Constraint at position level* and *Constraint at velocity level*.

1) Constraint at position level: Let $\mathbf{p_e} = [x_e \ y_e \ z_e]$ denote the translational components of $\mathbf{x_e} = [x_e \ y_e \ z_e \ \psi_e]$ (the current estimated pose) and Δt denote the control loop duration. Assuming Δt is small enough so the change in the yaw angle ψ is negligible, we have $\boldsymbol{p'} = \mathbf{p_e} + \Delta t \times \boldsymbol{v^{trans}}$ as the predicted position of the UAV in the next control loop (where $\boldsymbol{v^{trans}}$ contains translational components of $\boldsymbol{v}$). For other UAVs, let $\mathbf{p_e^i}$ be the current estimated positions of UAV $\mathbf{i}$ (recall that UAVs continually share their current positions). To satisfy the safe distance requirement, the distance between $\boldsymbol{p'}$ and each $\mathbf{p_e^i}$ must be larger than d_{safe}, that is:

$$||\boldsymbol{p'} - \mathbf{p_e^i}||_2 \geq d_{safe} \tag{7}$$

However, Equation (7) forms a non-linear and non-convex constraint which cannot be represented in a linear program. The non-convexity of Equation (7) can be seen in Figure 2. The blue dot represents the UAV we are modeling. There are two other UAVs, UAV1 and UAV2. To satisfy Equation (7), $\boldsymbol{p'}$ must be in the flyable zone (green region), which is non-convex.

To represent the safe distance requirement in a linear program, we linearize the constraint (7) at each control loop based on the current positions of the UAVs. For each UAV $\mathbf{i}$ of other UAVs, we create a plane P_i that has $\mathbf{n_i} = \mathbf{p_e} - \mathbf{p_e^i}$ as the normal vector and contains $\mathbf{p_e^i}$. Since the distance between $\boldsymbol{p'}$ and the plane P_i is always smaller than or equal to the distance

between $\boldsymbol{p'}$ and $\mathbf{p_e^i}$, we convert the constraint Equation (7) to the following linear constraint.

$$\frac{(\boldsymbol{p'} - \mathbf{p_e^i}) \cdot \mathbf{n_i}}{|\mathbf{n_i}|} \geq d_{safe} \tag{8}$$

The left hand side represents the distance between the point $\mathbf{p'}$ and the plane P_i. Constraint (8) states that the UAV must keep a distance of at least d_{safe} from plane P_i. Figure 3 illustrates constraint (8). Planes P_1 and P_2 are formed based on the current positions of the three UAVs in our example and the modeled UAV must keep a distance of at least d_{safe} from P_1 and P_2. The linearized constraints shape a conservative convex flyable zone. The planes P_i are recalculated at each time step based on the current positions of the UAVs.

Constraint (8) is a hard constraint and may lead to the constraint-infeasibility problem. Disturbances (such as errors in the dynamic model, localization and control command execution) could drive the UAV into a situation where the linear program has no feasible solution. For example, if the current distance between two UAVs are less than d_{safe}, there may exist no control command $\boldsymbol{u}$ so that constraint (8) is satisfied, that is, the safe distance cannot be achieved in the next time step.

To avoid the constraint-infeasibility problem, we soften the constraint (8) using a slack variable ε. The slack variable allows the constraint (8) to be violated and can be interpreted as the amount of violation of this constraint. We add an extra term $\omega \times \varepsilon$ to the objective function in order to reduce the amount of violation as much as possible, where ω is the weight of this constraint. The larger ω is, the more important to reduce the amount of constraint violation ε with respect other objective function terms such as the following desired trajectory term (Section IV-A, Equation (4)).

$$\begin{gathered} \frac{(\boldsymbol{p'} - \mathbf{p_e^i}) \cdot \mathbf{n_i}}{|\mathbf{n_i}|} \geq d_{safe} - \varepsilon \\ \varepsilon \geq 0 \\ \text{minimize } \omega \times \varepsilon \end{gathered} \tag{9}$$

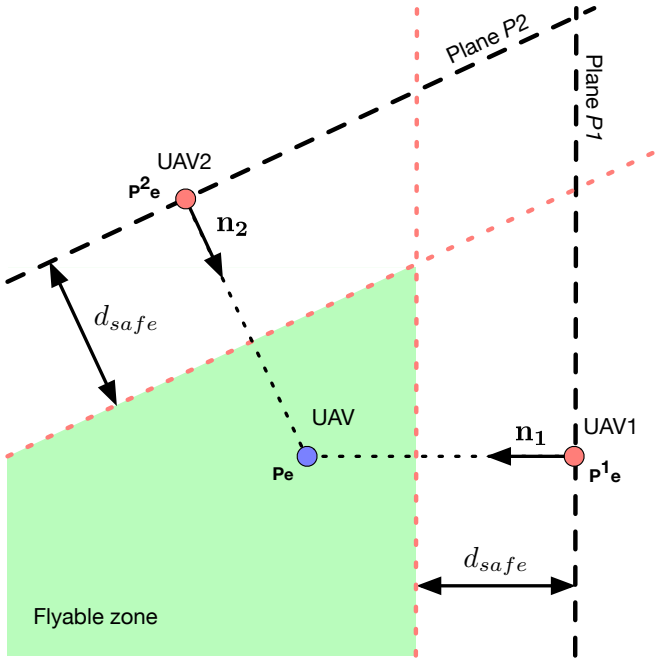


Fig. 3: Linearized safe distance constraint. The UAV (blue dot) needs to keep a distance of at least d_{safe} to planes P_1 (having normal vector $\mathbf{n_1}$) and P_2 (having normal vector $\mathbf{n_2}$). The flyable area forms a convex region.

2) Constraint at velocity level: The second way to model the safe distance requirement is to constrain the velocity of the UAV (instead of constraining its position as in the previous section).

Using the same linearization steps discussed above, we derive a plane P_i for each of other UAVs. Let $\mathbf{a_{max}}$ be a vector containing a conservative maximum translational acceleration of the UAV. $\mathbf{a_{max}}$ is a parameter of our constraint and it can be tuned. From $\mathbf{a_{max}}$, we calculate a^{de}_{max}, the maximum possible de-acceleration of the UAV away from the plane P_i, by projecting $\mathbf{a_{max}}$ on a vector containing the absolute values of the elements of the normal vector $\mathbf{n_i}$.

$$a^{de}_{max} = \left| \frac{\mathbf{a_{max}} \cdot abs(\mathbf{n_i})}{|\mathbf{n_i}|} \right| \tag{10}$$

Let $d = ||\mathbf{p_e} - \mathbf{p_e^i}||_2$ (the current distance between two UAVs). Given a^{de}_{max} and d, we compute v_{max}, the maximum scalar projection of the UAV's velocity on $-\mathbf{n_i}$ such that if we constantly apply a^{de}_{max} the UAV will fully stop moving towards the plane P_i at the distance of d_{safe} from the plane. If the safe distance is already violated at the current time step ($d < d_{safe}$), we set v_{max} using a proportional controller that steers the UAV away from the plane P_i.

$$v_{max} = \begin{cases} \sqrt{2 \times a^{de}_{max} \times (d - d_{safe})} & \text{if } d \geq d_{safe} \\ k_p \times (d - d_{safe}) & \text{if } d < d_{safe} \end{cases} \tag{11}$$

We then compute v^{next}_{max}, the conservative maximum projection of the velocity in the next time step.

$$v^{next}_{max} = v_{max} - a^{de}_{max} \times \Delta t \tag{12}$$

Finally, for each of other UAVs, we add a constraint saying that the next velocity must be bounded by v^{next}_{max}.

$$\begin{gathered} -\frac{\boldsymbol{v} \cdot \mathbf{n_i}}{|\mathbf{n_i}|} \leq v^{next}_{max} + \varepsilon \\ \varepsilon \geq 0 \\ \text{minimize } \omega \times \varepsilon \end{gathered} \tag{13}$$

The intuition behind constraint (13) is that the UAV must decrease the velocity component heading towards the UAV $\mathbf{i}$ when they are getting closer to each other. ω and ε have the same meanings as the ones in Equation (9).

There are several differences between constraining at position level and constraining at velocity level. Constraining at position level is simpler, more intuitive and more straightforward. However, the constraint only starts having effect

when the UAV is already close to the safe distance, that is, when the UAV is going to violate the safe distance in the next time step. Constraining at velocity level is less intuitive. However, it helps the UAV gradually decrease its velocity while approaching the safe distance, that is, the UAV reacts to the safe distance earlier. Note that, with the velocity-level constraint, only the velocity components heading toward other UAVs are constrained. The UAV can still fly at a high speed as long as those velocity components are limited.

C. Geofencing

We also apply the described safe distance constraints above to model the geofencing requirements. The only difference is that in the geofencing constraints, each geofence is already a plane given a priori and the safe distance $d_{safe} = 0$. Therefore, no linearization and position sharing is required.

D. Only move if visual localization is available

All the modeling techniques described above are general enough to be applied to different multi-rotor UAV platforms. The constraint described in this section is only applicable to Parrot Bebop UAVs.

Recall the safety requirement (3) in Section III, a UAV is only allowed to move if there is up-to-date visual localization information available. In our system, the estimated pose $\mathbf{x_e}$ is provided by visual localization (using a high resolution camera and artificial markers). The estimated velocity $\mathbf{v_e}$ is provided by the Parrot Bebop UAV's firmware independently, which is calculated by fusing data from an vertical optical flow camera and IMU, and is always available (we will discuss the system in detail in Section V-A). Therefore, even when the visual localization is lost, we still have the estimated velocity $\mathbf{v_e}$. That allows us to model this safety requirement by constraining the velocity $\boldsymbol{v}$ as follows.

$$-c\mathbf{M} \leq \boldsymbol{v} \leq c\mathbf{M} \tag{14}$$

where c is a constant equal to zero when there is no new estimated pose received after a duration (200 milliseconds in our case) and equal to one otherwise. The vector $\mathbf{M}$ contains positive values sufficiently larger than the maximum velocity of the UAV. In our application, since the estimated maximum velocity of the UAV is approximately $[3.5\ \ 3.5\ \ 3.5\ \ 10\pi/9]^T$, we select $\mathbf{M} = [10\ \ 10\ \ 10\ \ 10]^T$. When $c = 0$, $\boldsymbol{v}$ is enforced to be zero which makes the UAV hover. When $c = 1$, constraint Equation (14) does not have any effect on $\boldsymbol{v}$.

E. Control command

So far, the linear program is used to compute the velocity $\boldsymbol{v}$. In the remainder of this section we describe how to model the relation between $\boldsymbol{v}$ and the control command $\boldsymbol{u}$.

We model the control command $\boldsymbol{u}$ of the UAV as decision variables of the linear program. Different UAVs may require different control command $\boldsymbol{u}$. The Parrot Bebop UAVs accept the following control command:

$$\boldsymbol{u} = [\phi \quad \theta \quad \dot{z} \quad \dot{\psi}]^T \tag{15}$$

where ϕ, θ, $\dot{z}$, $\dot{\psi}$ are the pitch and roll angles, vertical velocity and yaw rotational velocity, respectively.

The following constraint represents the limits of the control command.

$$\mathbf{u_{min}} \leq \boldsymbol{u} \leq \mathbf{u_{max}} \tag{16}$$

$\boldsymbol{v}$ and $\boldsymbol{u}$ share the same $\dot{z}$ and $\dot{\psi}$ components (which means the same decision variables). Therefore, no extra constraint is needed to compute the $\dot{z}$ and $\dot{\psi}$ components of $\boldsymbol{u}$. To compute the pitch angle ϕ and roll angle θ components of $\boldsymbol{u}$ from the $\dot{x}$ and $\dot{y}$ components of $\boldsymbol{v}$, an acceleration controller and the dynamic model of the UAV are needed.

First, we define the acceleration $\boldsymbol{a^h} = [a_x\ \ a_y]^T$ of the UAV in the global reference frame as decision variables. Then, we add a proportional acceleration-controller representing the relation between $\boldsymbol{a^h}$ and $\boldsymbol{v}$ as a constraint.

$$\boldsymbol{a^h} = \mathbf{K_P^a}(\boldsymbol{v^h} - \mathbf{v_e^h}) \tag{17}$$

where $\boldsymbol{v^h}$ and $\mathbf{v_e^h}$ are the horizontal components (in x and y coordinates) of $\boldsymbol{v}$ and $\mathbf{v_e}$.

Since the pitch angle ϕ and roll angle θ components of $\mathbf{u}$ are in the body frame of the UAV, we need to derive the horizontal acceleration ${}_{\boldsymbol{b}}\boldsymbol{a^h}$ and velocity ${}_{\mathbf{b}}\mathbf{v_e^h}$ in the body frame. Given the current estimated yaw angle ψ_e, ${}_{\boldsymbol{b}}\boldsymbol{a^h}$ and ${}_{\mathbf{b}}\mathbf{v_e^h}$ are obtained through a simple transformation. The transformation is represented as the following constraints.

$${}_{\boldsymbol{b}}\boldsymbol{a^h} = \begin{bmatrix} cos(\psi_e) & -sin(\psi_e) \\ sin(\psi_e) & cos(\psi_e) \end{bmatrix} \boldsymbol{a^h} \tag{18}$$

$${}_{\mathbf{b}}\mathbf{v_e^h} = \begin{bmatrix} cos(-\psi_e) & -sin(-\psi_e) \\ sin(-\psi_e) & cos(-\psi_e) \end{bmatrix} \mathbf{v_e^h} \tag{19}$$

Then, ϕ and θ are derived from ${}_{\boldsymbol{b}}\boldsymbol{a^h}$ and ${}_{\mathbf{b}}\mathbf{v_e^h}$ using a simplified dynamic model of the UAV.

$${}_{\boldsymbol{b}}\boldsymbol{a^h} = -C \times {}_{\mathbf{b}}\mathbf{v_e^h} + G \times [\phi\ \ \theta]^T \tag{20}$$

The values of C and G depends on the UAV model. For example, the Parrot Bebop UAVs have:

$$C = [-0.576335778073963\ \ -0.584975281133000]^T$$
$$G = [9.81\ \ 9.81]^T$$

V. Validation

We validated our approach by deploying a prototype on real Parrot Bebop UAVs. In this section we report the validation result in our laboratory. Our approach was also applied to a real dance show with five UAVs performing on stage[2] (Figure 4).

[2]https://youtu.be/kMQtibTM2Lw

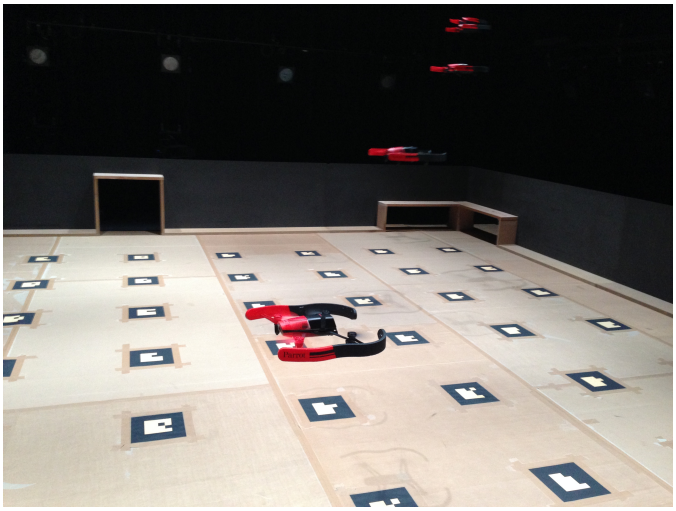


Fig. 4: Localization using AR markers. The Bebop's onboard flight controllers send font camera images and visual-inertial velocity to the group station computers. Pose estimation is done using front camera images and AR markers.

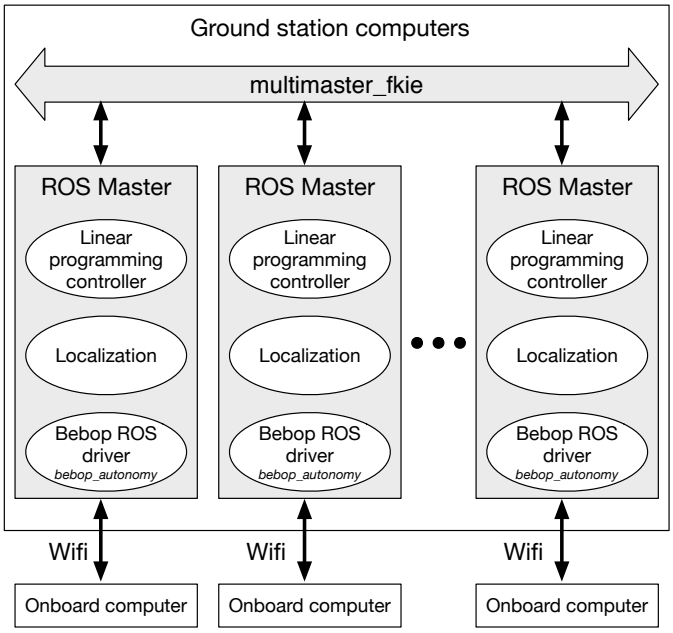


Fig. 5: System architecture. The onboard computer of each UAV communicates with ground station computers via Wifi. Each UAV is controlled by a ROS master. ROS master communicates with each other using the *multimaster_fkie* ROS package. The linear programming controller in each ROS master computes and sends control commands to the onboard flight controller.

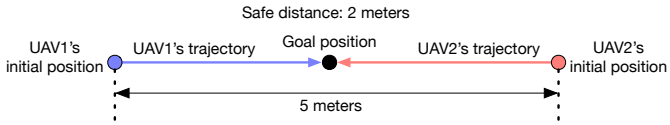


Fig. 6: Safe distance experiment setup. Two UAVs start at 5 meters away from each other and fly to the same goal position in X-Y plane. The safe distance is 2 meters. They fly at different heights to avoid being damaged if the software does not work as expected.

A. Implementation

Our software runs on two ground station computers with Ubuntu 14.04 and ROS Indigo. The ground stations communicate with Bebops' onboard computers via a 5 GHz WiFi link using the *bebop_autonomy*[3] ROS package. The onboard flight controllers send front camera images at a frequency of 24Hz and visual-inertial velocity (estimated by fusing data from the vertical optical flow camera and IMU) at a frequency of 5Hz. Pose estimation is done using the front camera images and AR markers (see Figure 4). We process the front camera images using OpenCV. Our localization software module interpolates the pose estimated using the front camera images and the visual-inertial velocity provided by Bebop's firmware. The visual localization using AR markers has the advantage of affordable price. Our tests show that this visual localization approach is robust in terms of pose error. However, the visual localization system is sensitive to the light condition. It could be the case that the light condition suddenly changes and the software cannot detect the markers anymore. That is why it is important to guarantee that the UAVs only move when they can detect markers and know their current poses.

The control is decentralized using a ROS multi-master system. Each UAV is controlled independently by a linear programming controller in a ROS master. The linear programming controller computes and sends control commands to the onboard flight controller. ROS masters communicate with each other using the *multimaster_fkie*[4] ROS package. Figure 5 describes our system architecture. Each UAV has its own ROS driver (*bebop_autonomy*), localization and controller components. The UAVs continually share their current pose with each other.

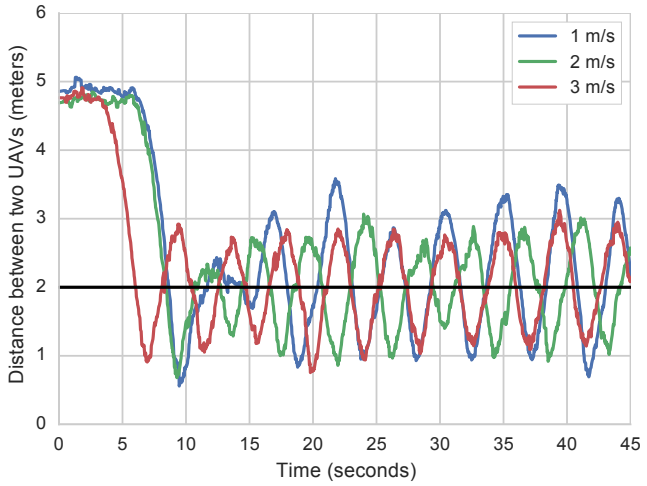
[3]http://wiki.ros.org/bebop_autonomy
[4]http://wiki.ros.org/multimaster_fkie

For all the following experiments, unless mentioned otherwise, we use the open-source linear programming solver Ojalgo[5] to solve our linear programs. The frequency of the control loop is 40Hz. Since in our scenarios, keeping safe distance and staying within geofence are more important than following desired trajectories, we set their weight $\omega = 1000$ empirically (see Equation (9) and (13)). Future work should focus on finding a systematic approach to set the weight of each constraint.

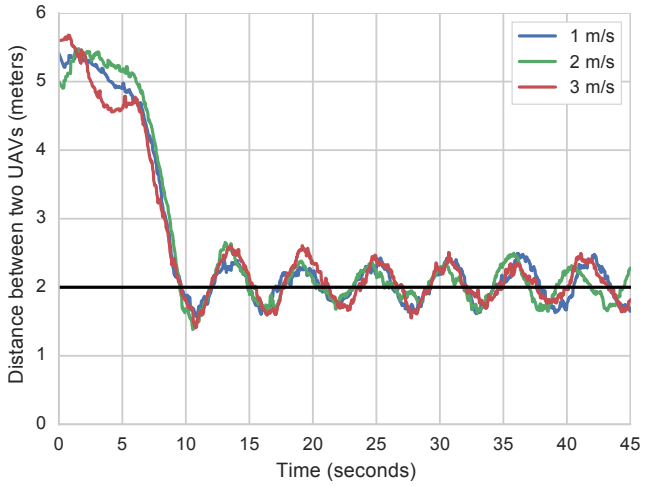
B. Safe distance experiment

The purpose of this experiment is to investigate how the UAVs react with the two techniques to model the safe distance requirement presented in Section IV-B. The experiment

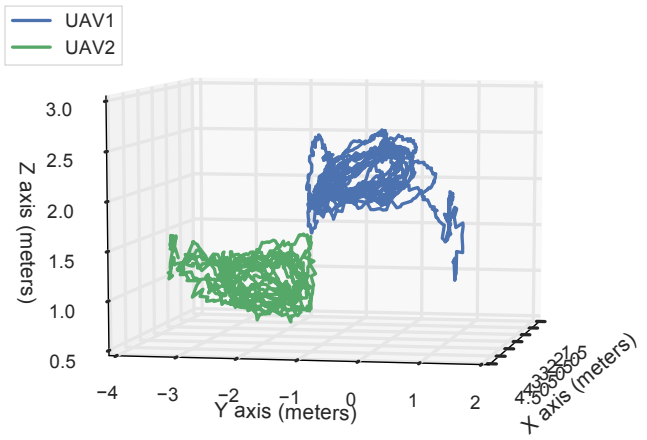
[5]http://ojalgo.org



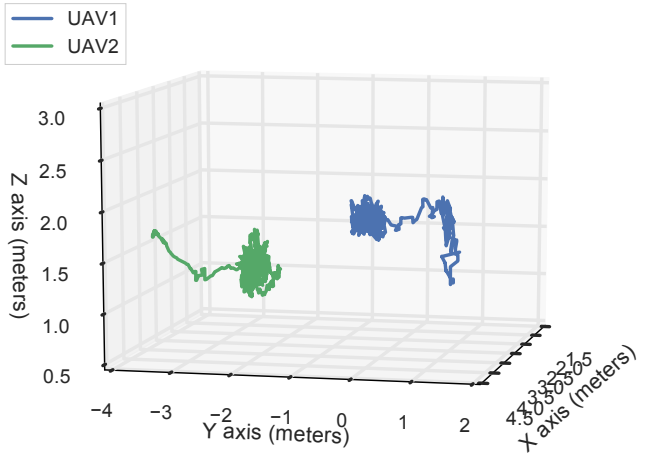
(a) Position-level constraint. The UAVs violate about 1 meter into the safe distance. The distance between two UAVs fluctuates significantly overtime.



(b) Velocity-level constraint with $\mathbf{a_{max}} = [0.2 \quad 0.2 \quad 0.2]^T$ (unit: m/s^2. The violated distance (about 0.5 meters) is less than that in the position-level constraint experiment. The distance between two UAVs is more stable.



(c) The movement of two UAVs in the position-level constraint experiment. They keep trying to move towards their common goal position and then aggressively flying further away from each other after violating the safe distance. Their movements spread over the space and are unpredictable.



(d) The movement of two UAVs in the velocity-level constraint experiment. When they are approaching the safe distance, they decrease their velocity and almost stay still at the safe distance. Their movements are stable and predictable.

Fig. 7: Safe distance experiment result. With the position level constraint, the UAVs cannot react to the safe distance when the UAVs fly at a high speed. With the velocity level constraint, the UAVs react more stable to the safe distance in the sense that the speed of the UAVs does not affect the amount of violated distance.

involves two UAVs. They initially start at 5 meters away from each other and fly toward the same goal position in X-Y plane. Their desired speed to follow the trajectories is a parameter of the experiment (1, 2 and 3 meters per second). The required safe distance is 2 meters. To avoid damaging the UAVs, we let them fly at different heights (1.25 meters and 1.75 meters). Figure 6 illustrates the 1D view of the setup. Figure 7 shows the result of the experiment.

Position-level constraint: Figure 7a and 7c show the resulting reaction of the UAVs when the safe distance requirement is modeled at position level. There are two conclusions from the experiment. First, the UAVs still violate the safe distance significantly (more than 1 meter). Because the constraint has effect on the movement behavior of the UAVs too late, they always overshoot and violate the safe distance. The violated distance depends on the flying speed and the agility of the UAVs. Second, the movement behavior of the UAVs is unstable. After violating the safe distance, the UAVs fly aggressively further away from each other. They then recover the safe distance, but then again try to fly towards their goal position and overshoot, and so on. That explains why in Figure 7a, the distance between the two UAVs fluctuates considerably over

time. The positions over time of the UAVs are unpredictable and spread over the space as shown in Figure 7c and in the video of the experiment[6].

Velocity-level constraint: Figure 7b and 7d show the result when the safe distance requirement is modeled at velocity level with $\mathbf{a_{max}} = [0.2 \quad 0.2 \quad 0.2]^T$ (unit: m/s^2). The first conclusion is that the UAVs still violate the safe distance. However, the violated distance (about 0.5 meters) is less than in the previous experiment with the position-level constraint. The reason is because by modeling safe distance at velocity level, the UAVs react earlier to the safe distance. They gradually decrease their speed when they are approaching each other as discussed in Section IV-B2. The second conclusion is that the movement behavior of the UAVs is more stable than in the position-level constraint experiment. They stay still at the safe distance from each other with the deviation of 0.5 meters. The positions overtime of the two UAVs in this experiment can be seen in Figure 7d and in the video of the experiment[7].

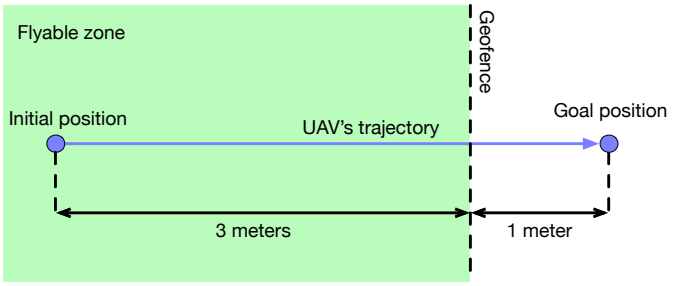


Fig. 8: Constraint tuning experiment setup

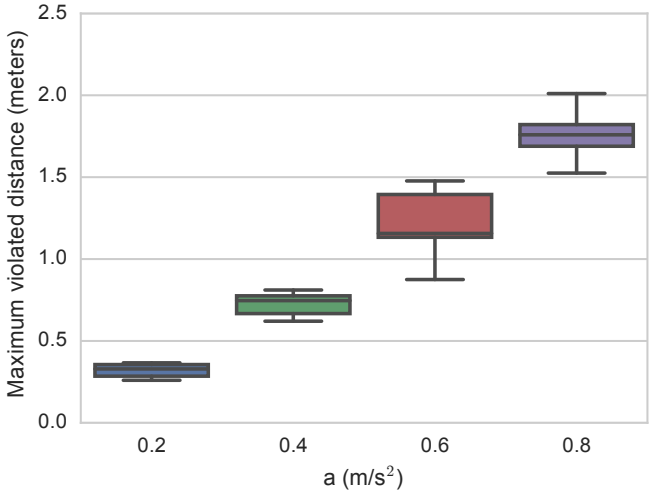


Fig. 9: Geofence violated distance with different $\mathbf{a_{max}} = [a \ a \ a]^T$ and with desired velocity of 3 m/s

C. Constraint tuning experiment

Due to uncertainty and error in the dynamic model, localization and control command execution, the velocity-level constraint cannot guarantee that the UAVs never violate the safe distance. However, it is our hypothesis that by tuning the parameter $\mathbf{a_{max}}$ (see Section IV-B2), we can control the violated distance and thus adapt the safe distance accordingly (by adding the maximum violated distance to the original safe distance) to achieve the hard safe distance requirement. In this experiment, we examine the movement behavior of the UAVs with different $\mathbf{a_{max}}$ values.

Figure 8 illustrates the experiment setup. A UAV starts from inside a flyable zone defined by a geofence and follows

[6]https://youtu.be/cNisyribY3Y
[7]See Footnote 6

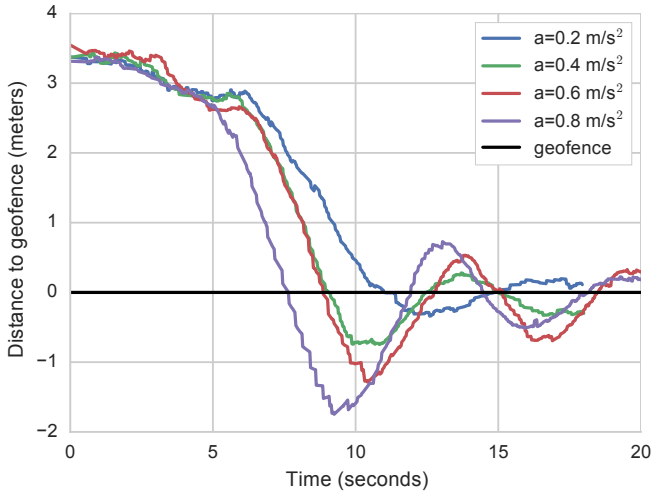


Fig. 10: Constraint tuning experiment result with different $\mathbf{a_{max}} = [a \ a \ a]^T$ and with desired velocity of 3 m/s

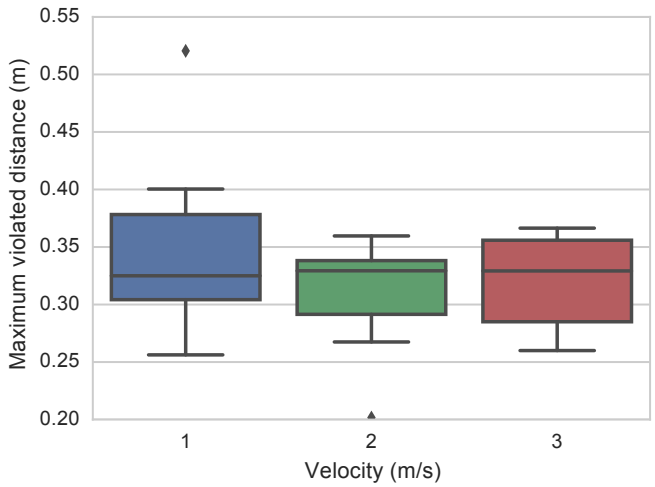


Fig. 11: Geofence violated distance with $\mathbf{a_{max}} = [0.2 \ 0.2 \ 0.2]^T$ and varied velocities

a trajectory with the goal position in the other side of the geofence (out of the flyable zone). The distance between the goal position and the geofence is 1 meter.

We let the UAV fly at the desired velocity of 3 meters per second and vary the values of $\mathbf{a_{max}}$. We measure the maximum violated distance to the geofence of the UAV. Each box plot in Figure 9 is the result of 10 trials. The result shows that if the values of $\mathbf{a_{max}}$ are high, the UAV is not able to stay within the geofence and may violate the geofence up to about 2 meters. It is because the limit over time of the velocity component heading toward the geofence plane depends on the values of $\mathbf{a_{max}}$. The lower $\mathbf{a_{max}}$ is, the lower the limit of the velocity component is. In other words, $\mathbf{a_{max}}$ helps limit the inertia of the UAV while it is flying towards a geofence (or another UAV), which in turn reduce the violated distance caused by overshooting. Figure 10 shows the distance over time from the UAV to the geofence with different $\mathbf{a_{max}}$ values. The distance is negative if the geofence is violated. With higher $\mathbf{a_{max}}$, the UAV overshoots more into the geofence.

Figure 11 shows the maximum violated distance when the UAV flies with different trajectory's desired velocities given $\mathbf{a_{max}} = [0.2\ 0.2\ 0.2]^T$. With an appropriate $\mathbf{a_{max}}$ value, the desired velocity does not affect the maximum violated distance. Therefore, we can empirically estimate the worst-case violated distance Δd and compute the new safe distance $d'_{safe} = d_{safe} + \Delta d$ to achieve the hard safe distance requirement. A too conservative $\mathbf{a_{max}}$ value will limit the desired movements of the UAV. Thus, the selection of $\mathbf{a_{max}}$ should depend on the scenario. Note that, in ideal scenarios without any uncertainty and error, we can derive the exact values for $\mathbf{a_{max}}$. However, such scenarios are unrealistic.

It is possible to use the variable a^{de}_{max} (see Section IV-B2) directly as a tuning variable instead of using $\mathbf{a_{max}}$. However, we prefer to use $\mathbf{a_{max}}$ as it has more straightforward kinematic meaning and it offers a higher degree of freedom in tuning. For example, it is better to use $\mathbf{a_{max}}$ when the maximum accelerations in the three translational dimensions are not the same.

The authors in [8] introduced a way to model the "keep a safe distance to obstacles" requirement as a least-squares problem. The difference between our model and the one in [8] is that our velocity-level constraint always actively takes into account the safe distance by limiting the velocity of the UAVs when they are approaching the geofence (or safe distance). With the model in [8], the UAV reacts to the safe distance in a passive way. Their constraint, which defines a proportional controller generating the desired velocity to push the UAV further away from obstacles, is only activated when the UAV already violated the safe distance to an obstacle. When the UAV does not violate the safe distance to any obstacle, the constraint is deactivated and has no effect on the behavior of the UAV. That way, the UAV may violate considerably into the safe distance due to its inertia while flying toward an obstacle. As yet there is no sensitivity analysis experiments reported in their work, it is unclear how well their model can keep the UAV at a safe distance to obstacles.

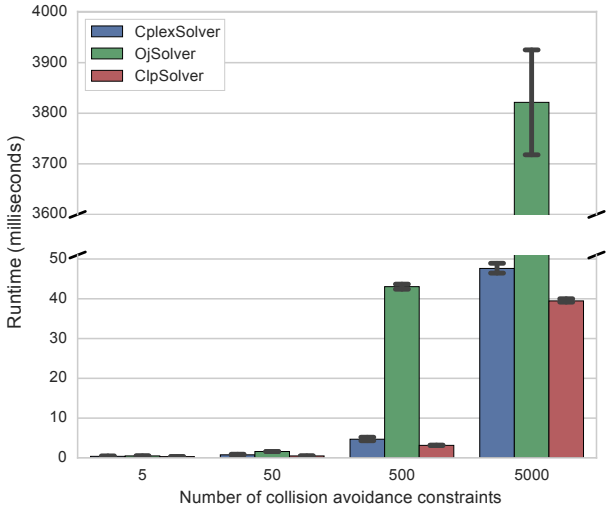


Fig. 12: Solver computational performance profiling

D. Computational performance profiling experiment

In this experiment we look into the scalability of our approach. We randomly generate linear programs presented in Section IV with artificial inputs and measure the time taken to solve those programs on an Ubuntu computer with Intel i7-7700K 4.20GHz processors. We use three linear programming solvers in this experiment: (1) CPLEX, a commercial solver, (2) Ojalgo, an open-source solver written in Java and (3) CLP[8] (Coin-or linear programming), an open-source solver written in C++. All the solvers use the Simplex method [21, Chapter 2] to solve our linear programs.

For each linear program, the following artificial inputs are generated: The current pose, desired pose, current velocity, desired velocity of the UAV and the current positions of other UAVs. The number of other UAVs is also the number of the safe distance constraints in the linear program. We perform the experiment with 5, 50, 500 and 5000 safe distance constraints. For each number of constraints, 1000 linear programs are generated.

Since we execute the linear programming controller at a frequency of 40Hz, it should take less than 25 milliseconds to solve a linear program. The result in Figure 12 shows that with 5 and 50 safe distance constraints, the solving time of all solvers is insignificant (only few milliseconds). With 500 constraints, CPLEX and CLP can still solve a linear program in less than 25 milliseconds but Ojalgo needs around 45 milliseconds on average. With 5000 constraints, CPLEX needs about 50 milliseconds, CLP needs about 40 milliseconds and Ojalgo needs nearly 4 seconds to solve a linear program.

The result shows that our approach is feasible even when the number of UAVs is large. The approach can work with more than 500 UAVs using both open-source (CLP) and commercial (CPLEX) solvers. Future research should focus on reducing

[8]https://projects.coin-or.org/Clp

the runtime of the solver (by, for example, removing redundant constraints).

Since in our dance show only five UAVs are used, the open-source solver Ojalgo is sufficient and is used as the solver of all our linear programming controllers. We did not use CPLEX due to licensing restrictions and we were not aware of CLP when we developed the dance show.

VI. Conclusions

In this paper we present the use of linear programing in developing software for indoor choreographed UAVs. Although some real-world constraints are non-linear, it is possible to approximate them as linear constraints and objectives, taking the advantage of the computational efficiency of linear programming.

Formulating the movement behavior of UAVs as a linear program enables a way to integrate safety requirements into a single control software. However, care must be taken when modeling safety requirements so that the resulting reaction of the UAVs is stable and predictable.

We proposed two techniques to model the safe distance and geofencing requirements. Our experiments on real UAVs show that the velocity-level constraint results in more stable behavior than using the position-level constraint. The velocity-level constraint also allows us to control the violated distance without regard to the desired flying speed of the UAVs, enabling us to achieve the hard safe distance and geofencing by empirically tuning parameters and estimating the worst-case violated distance. Our profiling experiment shows that our approach can be deployed using existing open-source and commercial linear programming solvers.

Our future work will focus on extending our approach so that it can take into account safety concerns in motion planning and other long-term reasoning requirements.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven, and by the imec-ICON project SafeDroneWare. We thank Marvin Ferber and Kristof Coninx for helping us develop our system. We are grateful to the anonymous reviewers for their helpful comments.

References

[1] A. Schllig, M. Hehn, S. Lupashin, and R. D'Andrea, "Feasiblity of motion primitives for choreographed quadrocopter flight," in *American Control Conference (ACC), 2011*. IEEE, 2011, pp. 3843–3849.

[2] F. Augugliaro, A. P. Schoellig, and R. DAndrea, "Methods for Designing and Executing an Aerial Dance Choreography," 2013.

[3] A. P. Schoellig, H. Siegel, F. Augugliaro, and R. DAndrea, "So You Think You Can Dance? Rhythmic Flight Performances with Quadrocopters," in *Controls and Art*. Springer, 2014, pp. 73–105.

[4] E. Cappo, A. Desai, and N. Michael, "Robust Coordinated Aerial Deployments for Theatrical Applications Given Online User Interaction via Behavior Composition," in *13th International Symposium on Distributed Autonomous Robotic Systems*, 2016.

[5] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decr, R. Smits, E. Aertbelin, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

[6] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks," *IEEE Transactions on Robotics*, vol. 25, no. 3, pp. 670–685, 2009.

[7] A. M. Zanchettin and P. Rocco, "Reactive motion planning and control for compliant and constraint-based task execution," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 2748–2753.

[8] J. Verbeke, J. Vantilt, D. Vanthienen, M. Vochten, S. Debruyne, and J. De Schutter, "A constraint-based flight control system architecture for UAVs using the iTaSC framework," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2016, pp. 310–319.

[9] E. Simetti and G. Casalino, "A novel practical technique to integrate inequality control objectives and task transitions in priority based control," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1-4, pp. 877–902, 2016.

[10] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 42–49.

[11] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, 2016.

[12] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization." in *Robotics: science and systems*, vol. 9. Citeseer, 2013, pp. 1–10.

[13] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[14] M. Abdolhosseini, Y. M. Zhang, and C. A. Rabbath, "An efficient model predictive control scheme for an unmanned quadrotor helicopter," *Journal of intelligent & robotic systems*, pp. 1–12, 2013.

[15] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic systems architectures and programming," in *Springer Handbook of Robotics*. Springer, 2016, pp. 283–306.

[16] W. Decre, R. Smits, H. Bruyninckx, and J. D. Schutter, "Extending iTaSC to support inequality constraints and non-instantaneous task specification," in *IEEE International Conference on Robotics and Automation, 2009. ICRA '09*, May 2009, pp. 964–971.

[17] W. Decre, H. Bruyninckx, and J. De Schutter, "Extending the iTaSC constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1941–1948.

[18] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. wiley New York, 2006, vol. 3.

[19] P.-B. Wieber, R. Tedrake, and S. Kuindersma, "Modeling and control of legged robots," in *Springer Handbook of Robotics*. Springer, 2016, pp. 1203–1234.

[20] C. Powers, D. Mellinger, and V. Kumar, "Quadrotor Kinematics and Dynamics," in *Handbook of Unmanned Aerial Vehicles*. Springer, 2015, pp. 307–328.

[21] R. J. Vanderbei, *Linear programming*. Springer, 2015.